TPU-MLIR快速入门指南

> 文档版本: 0.1.0
>
> 发布日期: 2022-07-28
>
> 适用于 BM1684x芯片

目录

法律声明

本数据手册包含北京算能科技有限公司（下称"算能"）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致算能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

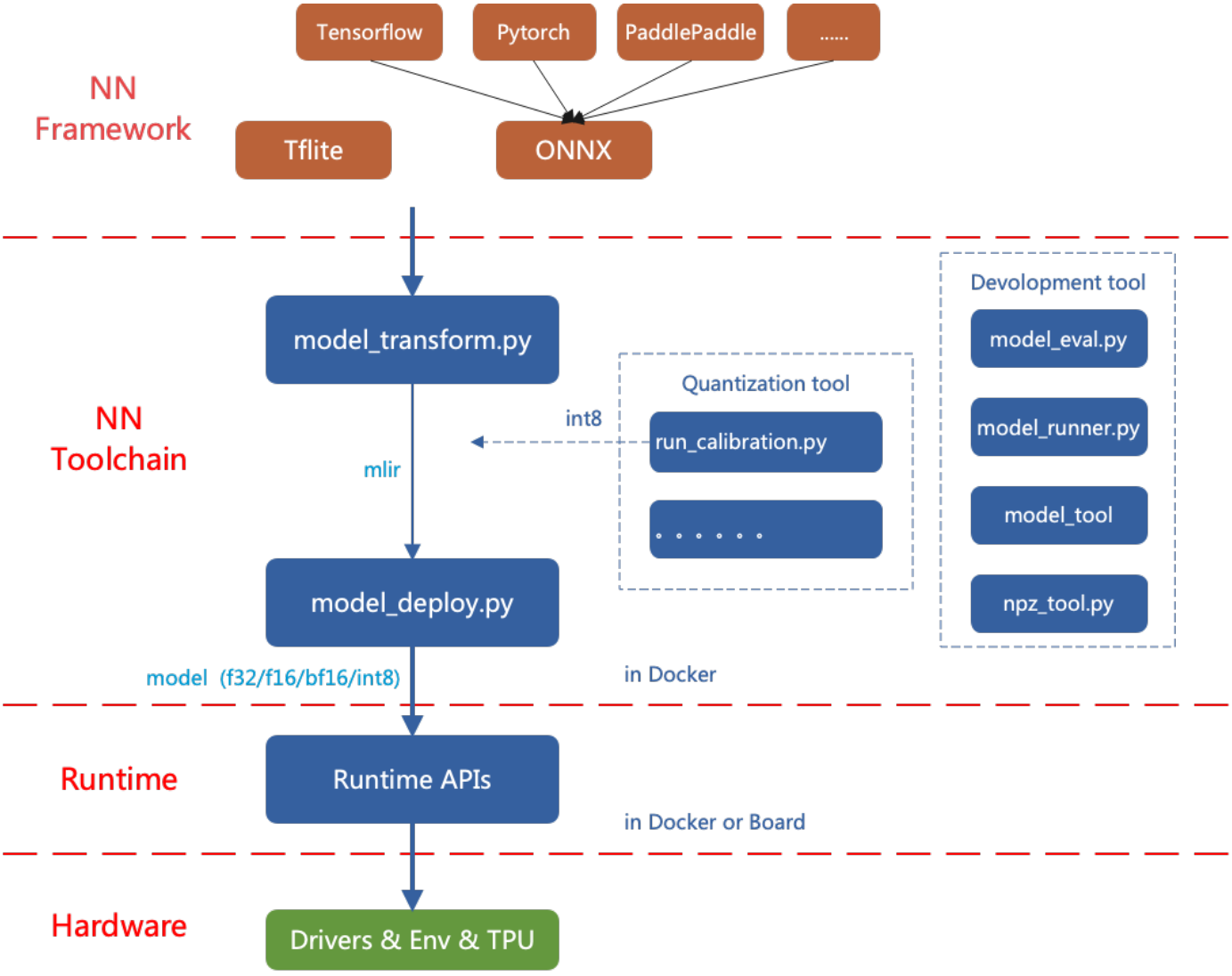
本文件内信息如有更改，恕不另行通知。算能不对使用或依赖本文件所含信息承担任何责任。

本数据手册和本文件所含的所有信息均按"原样"提供，无任何明示、暗示、法定或其他形式的保证。算能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，算能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

1 TPU-MLIR简介

TPU-MLIR是算能智能AI芯片的TPU编译器工程。该工程提供了一套完整的工具链，其可以将不同框架下预训练的神经网络，转化为可以在算能TPU上高效运算的二进制文件`bmodel`。代码已经开源到github: https://github.com/sophgo/tpu-mlir

TPU-MLIR的整体架构如下：



目前直接支持的框架有tflite和onnx。其他框架的模型需要转换成onnx模型。如何将其他深度学习架构的网络模型转换成onnx，可以参考onnx官网: https://github.com/onnx/tutorials

转模型需要在指定的docker执行，主要分两步，一是通过`model_transform.py`将原始模型转换成mlir文件，二是通过`model_deploy.py`将mlir文件转换成bmodel。如果要转INT8模型，则需要调用`run_calibration.py`生成量化表传给`model_deploy.py`。本文主要就是介绍这个模型转换的过程。

2 开发环境配置

从DockerHub:https://hub.docker.com/r/sophgo/tpuc_dev下载所需的镜像:

```
docker pull sophgo/tpuc_dev:v1.1
```

如果是首次使用docker，可执行下述命令进行安装和配置（仅首次执行）：

```
sudo apt install docker.io
sudo systemctl start docker
sudo systemctl enable docker

sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker (use before reboot)
```

确保安装包在当前目录，然后在当前目录创建容器如下：

```
# myname只是举个名字的例子，请指定成自己想要的容器的名字
docker run --privileged --name myname -v $PWD:/workspace -it sophgo/tpuc_dev:v1.1
```

后文假定用户已经处于docker里面的/workspace目录。

3 编译onnx模型

本章以`yolov5s.onnx`为例，介绍如何编译迁移一个onnx模型至BM1684x TPU平台运行。

该模型来在yolov5的官网: https://github.com/ultralytics/yolov5/releases/download/v6.0/yolov5s.onnx

本章需要如下文件(其中xxxx对应实际的版本信息):

- tpu-mlir_xxxx.tar.gz (tpu-mlir的发布包)

步骤 0：加载tpu-mlir

```
tar zxf tpu-mlir_xxxx.tar.gz
source tpu-mlir_xxxx/envsetup.sh
```

步骤 1：准备工作目录

建立`model_yolov5s`目录，注意是与tpu-mlir同级目录；并把模型文件和图片文件都放入`model_yolov5s`目录中。

操作如下:

```
mkdir model_yolov5s && cd model_yolov5s
cp $TPUC_ROOT/regression/model/yolov5s.onnx .
cp -rf $TPUC_ROOT/regression/dataset/COCO2017 .
cp -rf $TPUC_ROOT/regression/image .
mkdir workspace && cd workspace
```

这里的`$TPUC_ROOT`是环境变量，对应tpu-mlir_xxxx目录

步骤 2：模型转MLIR

如果模型是图片输入，在转模型之前我们需要了解模型的预处理。如果模型用预处理后的npz文件做输入，则不需要考虑预处理。
预处理过程用公式表达如下（x代表输入):

$$y = （x - mean）\times scale$$

官网yolov5的图片是rgb，每个值会乘以`1/255`，转换成mean和scale对应为`0.0,0.0,0.0`和`0.0039216,0.0039216,0.0039216`。

模型转换命令如下:

```
model_transform.py \
    --model_name yolov5s \
    --model_def ../yolov5s.onnx \
    --input_shapes [[1,3,640,640]] \
    --mean 0.0,0.0,0.0 \
    --scale 0.0039216,0.0039216,0.0039216 \
    --keep_aspect_ratio \
    --pixel_format rgb \
    --output_names 350,498,646 \
    --test_input ../image/dog.jpg \
    --test_result yolov5s_top_outputs.npz \
    --mlir yolov5s.mlir
```

`model_transform.py`支持的参数如下:

参数名	必选?	说明
model_name	是	指定模型名称
model_def	是	指定输入文件用于验证，可以是图片或npy或npz；可以不指定，则不会正确性验证
input_shapes		指定输入的shape，例如[[1,3,640,640]]；二维数组，可以支持多输入情况
resize_dims		原始图片需要resize之后的尺寸；如果不指定，则resize成模型的输入尺寸
keep_aspect_ratio		在Resize时是否保持长宽比，默认为false；设置时会对不足部分补0
mean		图像每个通道的均值，默认为0.0,0.0,0.0
scale		图片每个通道的比值，默认为1.0,1.0,1.0
pixel_format		图片类型，可以是rgb、bgr、gray、rgbd四种情况
output_names		指定输出的名称，如果不指定，则用模型的输出；指定后用该指定名称做输出
test_input		指定输入文件用于验证，可以是图片或npy或npz；可以不指定，则不会正确性验证
test_result		指定验证后的输出文件
excepts		指定需要排除验证的网络层的名称，多个用,隔开
mlir	是	指定输出的mlir文件路径

转成mlir文件后，会生成一个`${model_name}_in_f32.npz`文件，该文件是模型的输入文件。它是通过对图片输入进行预处理后得到的数据。

步骤 3：MLIR转F32模型

将mlir文件转换成f32的bmodel，操作方法如下：

```
model_deploy.py \
  --mlir yolov5s.mlir \
  --quantize F32 \
  --chip bm1684x \
  --test_input yolov5s_in_f32.npz \
  --test_reference yolov5s_top_outputs.npz \
  --tolerance 0.99,0.99 \
  --model yolov5s_1684x_f32.bmodel
```

model_deploy.py的相关参数说明如下：

参数名	必选?	说明
mlir	是	指定mlir文件
quantize	是	指定默认量化类型，支持F32/BF16/F16/INT8
chip	是	指定模型将要用到的平台，支持bm1684x（目前只支持这一种，后续会支持多款TPU平台）
calibration_table		指定量化表路径，当存在INT8量化的时候需要量化表
tolerance		表示 MLIR 量化后的结果与 MLIR fp32推理结果相似度的误差容忍度
correctnetss		表示仿真器运行的结果与MLIR量化后的结果相似度的误差容忍度，默认0.99,0.99
excepts		指定需要排除验证的网络层的名称，多个用,隔开
model	是	指定输出的model文件路径

步骤 4：MLIR转INT8模型

转INT8模型前需要跑calibration，得到量化表；输入数据的数量根据情况准备100~1000张左右。

这里用现有的100张来自COCO2017的图片举例，执行calibration：

```
run_calibration.py yolov5s.mlir \
  --dataset ../COCO2017 \
  --input_num 100 \
  -o yolov5s_cali_table
```

转成INT8对称量化模型，执行如下命令：

```
model_deploy.py \
  --mlir yolov5s.mlir \
  --quantize INT8 \
  --calibration_table yolov5s_cali_table \
  --chip bm1684x \
  --test_input yolov5s_in_f32.npz \
  --test_reference yolov5s_top_outputs.npz \
  --tolerance 0.88,0.50 \
  --correctness 0.99,0.90 \
  --model yolov5s_1684x_int8_sym.bmodel
```

步骤5：效果对比

在本发布包中有用python写好的yolov5用例，源码路径`$TPUC_ROOT/python/samples/detect_yolov5.py`，用于对图片进行目标检测。阅读该代码可以了解模型是如何使用的。以下用该程序分别来查看onnx/f32/int8的执行结果。

onnx模型的执行方式如下，得到`dog_onnx.jpg`：

```
detect_yolov5.py \
  --input ../image/dog.jpg \
  --model ../yolov5s.onnx \
  --output dog_onnx.jpg
```

f32 bmodel的执行方式如下，得到`dog_f32.jpg`：

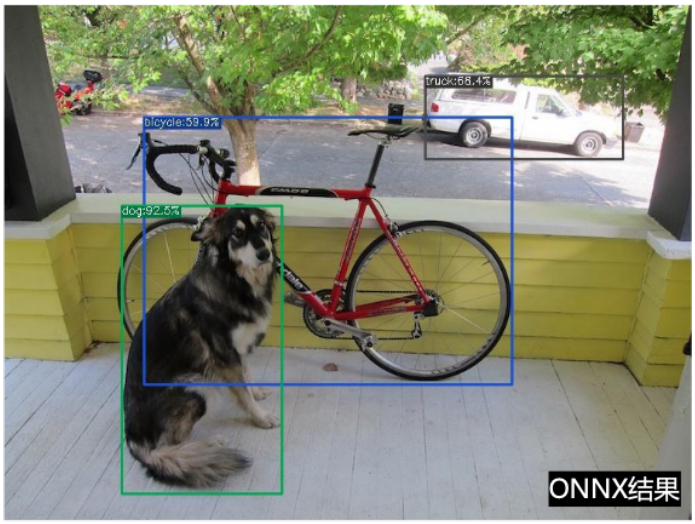
```
detect_yolov5.py \
  --input ../image/dog.jpg \
  --model yolov5s_1684x_f32.bmodel \
  --output dog_f32.jpg
```

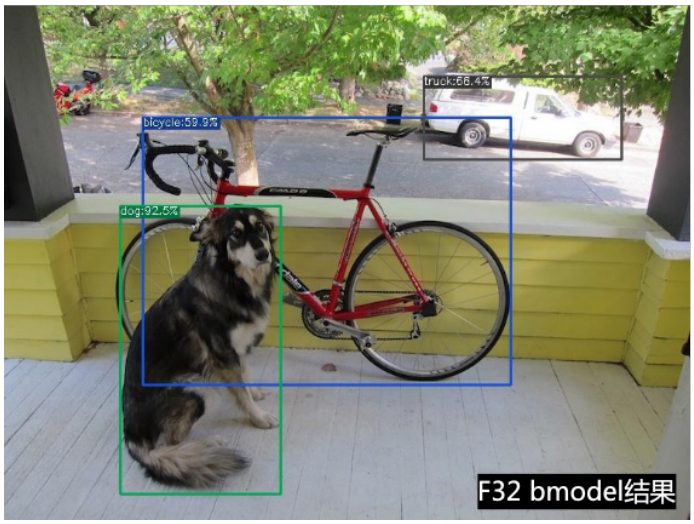
int8 bmodel的执行方式如下，得到dog_int8_sym.jpg：

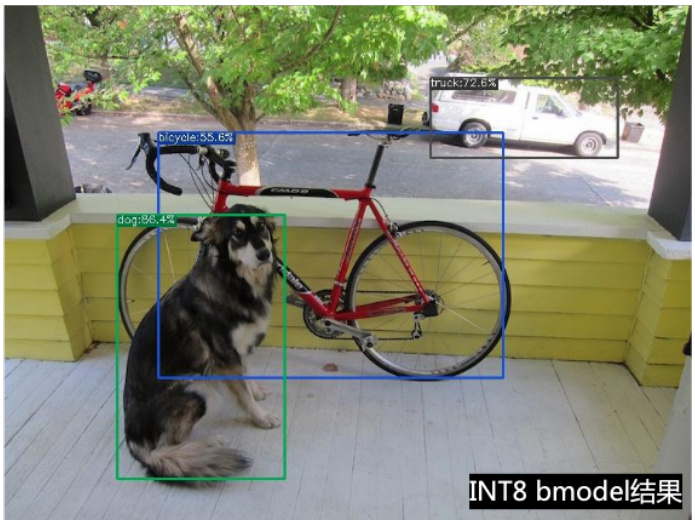
```
detect_yolov5.py \
  --input ../image/dog.jpg \
  --model yolov5s_1684x_int8_sym.bmodel \
  --output dog_int8_sym.jpg
```

四张图片对比如下：


原始图片


truck:68.4%
bicycle:59.9%
dog:92.5%
ONNX结果


truck:68.4%
bicycle:59.9%
dog:92.5%
F32 bmodel结果


truck:72.6%
bicycle:55.6%
dog:86.4%
INT8 bmodel结果